Low-Complexity RLS Algorithms Using Dichotomous Coordinate Descent Iterations

Yuriy V. Zakharov, *Member, IEEE*, George P. White, and Jie Liu

***Abstract*—In this paper, we derive low-complexity recursive least squares (RLS) adaptive filtering algorithms. We express the RLS problem in terms of auxiliary normal equations with respect to increments of the filter weights and apply this approach to the exponentially weighted and sliding window cases to derive new RLS techniques. For solving the auxiliary equations, line search methods are used. We first consider conjugate gradient iterations with a complexity of $\mathcal{O}(N^2)$ operations per sample; N being the number of the filter weights. To reduce the complexity and make the algorithms more suitable for finite precision implementation, we propose a new dichotomous coordinate descent (DCD) algorithm and apply it to the auxiliary equations. This results in a transversal RLS adaptive filter with complexity as low as $3N$ multiplications per sample, which is only slightly higher than the complexity of the least mean squares (LMS) algorithm ($2N$ multiplications). Simulations are used to compare the performance of the proposed algorithms against the classical RLS and known advanced adaptive algorithms. Fixed-point FPGA implementation of the proposed DCD-based RLS algorithm is also discussed and results of such implementation are presented.**

***Index Terms*—Adaptive filter, conjugate gradient, DCD algorithm, dichotomous coordinate descent, FPGA implementation, line search, RLS.**

I. Introduction

In adaptive filtering, the recursive least squares (RLS) algorithm is known to possess fast convergence, but also to have a high complexity of $\mathcal{O}(N^2)$ operations per sample (N being the filter length) [1], [2]. There has been much research interest in the reduction in complexity of the RLS algorithm. It is desirable to find a solution that has complexity similar to that of the least mean squares (LMS) algorithm, i.e., $2N$ multiplications per sample. In [2], fast RLS algorithms are summarized in terms of complexity. The *fixed-order* adaptive filters, exploiting the shifted structure of data vectors, have a complexity of $\mathcal{O}(N)$. The fastest among them in terms of multiplications is the fast Kalman filter that requires $6N$ multiplications [2]. The fixed-order algorithms suffer from numerical instability in finite precision implementation. This problem is partly overcome by using stabilization techniques. However, these make the algorithms more complicated, and, even with such techniques, they can still exhibit instability [2]. Another group of fast adaptive algorithms is the *lattice algorithms*. However, lattice algorithms do not provide the filter weights required in many applications, and their complexity is still high; the techniques considered in [2] require at least $20N$ multiplications and divisions per sample. Recently, the KaGE RLS algorithm was introduced [3]; it uses the shifted structure of data vectors, generates the filter weights, and its complexity is $\mathcal{O}(N \log_2 N)$, more specifically, $13N \log_2 N$ multiplications per sample. However, the KaGE algorithm also requires $\mathcal{O}(N \log_2 N)$ divisions.

Many adaptive algorithms require division and square root operations, which are complex for implementation, especially in hardware, i.e., they require a significant chip area and high power consumption. Although simpler than divisions, multiplications are still significantly more difficult for implementation than additions. Therefore, it is important to design algorithms that have no division, no square root operations, and as few multiplications as possible.

Many fast adaptive algorithms are based on matrix inversion which results in instability in finite precision implementation. An alternative approach based on solving the normal equations [4] often results in stable adaptive algorithms. Such an approach is used in the *direction set* (or *line search*) based adaptive algorithms. These techniques have either a good RLS-like performance but a high complexity of $\mathcal{O}(N^2)$, e.g., the conjugate gradient [5]–[8] or Euclidean direction search (EDS) [9] adaptive algorithms, or a low complexity of $\mathcal{O}(N)$ but a low performance, e.g., the fast EDS algorithm [9]–[11] or the stochastic line search algorithm [12].

The contributions of this paper are as follows:

1) A new formulation of the RLS problem in terms of a sequence of auxiliary normal equations with respect to increments of the filter weights is proposed (Section II). This formulation, though simple, results in adaptive filtering algorithms with high performance.
2) Two new general structures of adaptive filters for the exponentially weighted and sliding window RLS problems are introduced (Sections II-A and -B, respectively), and further specified for transversal adaptive filters (Section II-C).
3) 3) New RLS adaptive filtering algorithms based on conjugate gradient (CG) and coordinate descent (CD) iterations are proposed (Sections III-A and -B, respectively), whose particular implementations correspond to known adaptive algorithms.
4) A new dichotomous coordinate descent (DCD) algorithm is proposed (Section III-C).
5) New DCD-based RLS algorithms, in particular, DCD-based transversal RLS algorithms with complexity as low as $3N$ multiplications per sample are proposed.

Manuscript received June 25, 2007; revised December 17, 2007. The associate editor coordinating the review of this manuscript and approving it for publication was Dr. Alper Tunga Erdogan.

Y. V. Zakharov and J. Liu are with the Department of Electronics, University of York, York YO10 5DD, U.K. (e-mail: yz1@ohm.york.ac.uk; jl512@ohm.york.ac.uk).

G. P. White is with Communications Division, QinetiQ Ltd., Malvern, WR14 3PS, Worcestershire, U.K. (e-mail: gpwhite@qinetiq.com).

Digital Object Identifier 10.1109/TSP.2008.917874

6) FPGA design of DCD-based adaptive RLS algorithms is described, that significantly outperforms FPGA designs of known RLS algorithms, such as the QRD-RLS algorithm (Section IV).

The rest of the paper is organized as follows. In Section IV, we consider practical issues of implementing the proposed algorithms. Section V presents simulation results that show the performance of the proposed algorithms against the classical RLS algorithm and other known adaptive algorithms. Finally, Section VI gives conclusions.

A part of the material of this paper, namely those related to the exponentially weighted DCD-based RLS algorithm, was presented at the conference Asilomar 2007.

Notations: In this paper, we use capital and small bold fonts to denote matrices and vectors, respectively; e.g., $\mathbf{R}$ and $\mathbf{r}$. Elements of the matrix and vector are denoted as $R_{p,n}$ and r_n. $\mathbf{R}^T$ denotes transpose of $\mathbf{R}$. A pth column of $\mathbf{R}$ is denoted as $\mathbf{R}^{(p)}$. The variable i is used as a time index, i.e., $\mathbf{R}(i)$ is the matrix $\mathbf{R}$ at time instant i. The variable k is used as an iteration index. Only real-valued adaptive filtering is considered in this paper; the extension to the complex-valued case is straightforward.

II. Problem Statement and Recursive Solution of the RLS Normal Equations

In the RLS problem, at every time instant $i (i = 0, 1, 2, \ldots)$, an adaptive algorithm should find a solution to the normal equations

$$\mathbf{R}(i)\mathbf{h}(i) = \boldsymbol{\beta}(i) \tag{1}$$

where $\mathbf{R}(i)$ is assumed to be a symmetric positive-definite (correlation) matrix of size $N \times N, \boldsymbol{\beta}(i)$ and $\mathbf{h}(i)$ are N- length vectors. The matrix $\mathbf{R}(i)$ and vector $\boldsymbol{\beta}(i)$ are known, whereas the vector $\mathbf{h}(i)$ should be estimated. Direct methods for solving the system are too complex for most applications of adaptive filtering, especially if N is high; e.g., the Cholesky decomposition finds the solution with a complexity $\mathcal{O}(N^3)$ [13]. In the classical RLS algorithm, the solution is represented in the form [2]: $\mathbf{h}(i) = \mathbf{P}(i)\boldsymbol{\beta}(i)$, where $\mathbf{P}(i) = \mathbf{R}^{-1}(i)$; $\mathbf{P}(i)$ can be computed recursively with a complexity of $\mathcal{O}(N^2)$ [2]. We adopt another approach, which is based on transforming the original sequence of normal equations (1) into a sequence of auxiliary normal equations that are then solved by using iterative techniques.

Let, at time instant $(i-1)$, a system of equations $\mathbf{R}(i-1)\mathbf{h}(i-1) = \boldsymbol{\beta}(i-1)$ be approximately solved, and the approximate solution is $\hat{\mathbf{h}}(i-1)$. Let

$$\mathbf{r}(i-1) = \boldsymbol{\beta}(i-1) - \mathbf{R}(i-1)\hat{\mathbf{h}}(i-1) \tag{2}$$

be a residual vector for this solution. At time instant i, the system (1) is to be solved. We denote $\Delta\mathbf{R}(i) = \mathbf{R}(i) - \mathbf{R}(i-1), \Delta\boldsymbol{\beta}(i) = \boldsymbol{\beta}(i) - \boldsymbol{\beta}(i-1)$, and

$$\Delta\mathbf{h}(i) = \mathbf{h}(i) - \hat{\mathbf{h}}(i-1). \tag{3}$$

Our aim is to find a solution $\hat{\mathbf{h}}(i)$ of (1) by exploiting the previously obtained solution $\hat{\mathbf{h}}(i-1)$ and residual vector $\mathbf{r}(i-1)$. Equation (1) can be rewritten as

$$\mathbf{R}(i)[\hat{\mathbf{h}}(i-1) + \Delta\mathbf{h}(i)] = \boldsymbol{\beta}(i) \tag{4}$$

TABLE I
Recursively Solving a Sequence of Systems of Equations

Step	Equation
	Initialization: $\mathbf{r}(-1) = \mathbf{0}, \boldsymbol{\beta}(-1) = \mathbf{0}, \hat{\mathbf{h}}(-1) = \mathbf{0}$
	for $i = 0, 1, \ldots$
1	Find $\Delta\boldsymbol{R}(i)$ and $\Delta\boldsymbol{\beta}(i)$
2	$\boldsymbol{\beta}_0(i) = \mathbf{r}(i-1) + \Delta\boldsymbol{\beta}(i) - \Delta\mathbf{R}(i)\hat{\mathbf{h}}(i-1)$
3	Solve $\mathbf{R}(i)\Delta\mathbf{h} = \boldsymbol{\beta}_0(i) \Rightarrow \Delta\hat{\mathbf{h}}(i), \mathbf{r}(i)$
4	$\hat{\mathbf{h}}(i) = \hat{\mathbf{h}}(i-1) + \Delta\hat{\mathbf{h}}(i)$

and represented as a system of equations with respect to the unknown vector $\Delta\mathbf{h}(i)$

$$\begin{aligned}\mathbf{R}(i)&\Delta\mathbf{h}(i)\\ &= \boldsymbol{\beta}(i) - \mathbf{R}(i)\hat{\mathbf{h}}(i-1)\\ &= \boldsymbol{\beta}(i) - \mathbf{R}(i-1)\hat{\mathbf{h}}(i-1) - \Delta\mathbf{R}(i)\hat{\mathbf{h}}(i-1)\\ &= \mathbf{r}(i-1) + \Delta\boldsymbol{\beta}(i) - \Delta\mathbf{R}(i)\hat{\mathbf{h}}(i-1).\end{aligned} \tag{5}$$

Instead of solving the original problem (1), one can find a solution $\Delta\hat{\mathbf{h}}(i)$ of the auxiliary system of equations

$$\mathbf{R}(i)\Delta\mathbf{h}(i) = \boldsymbol{\beta}_0(i) \tag{6}$$

where

$$\boldsymbol{\beta}_0(i) = \mathbf{r}(i-1) + \Delta\boldsymbol{\beta}(i) - \Delta\mathbf{R}(i)\hat{\mathbf{h}}(i-1) \tag{7}$$

and obtain an approximate solution of the original system (1) as

$$\hat{\mathbf{h}}(i) = \hat{\mathbf{h}}(i-1) + \Delta\hat{\mathbf{h}}(i). \tag{8}$$

It is seen from (7) that this approach requires the residual vector $\mathbf{r}(i)$ for the solution $\hat{\mathbf{h}}(i)$ to the original system (1) to be known at each time instant i. After some algebra, we obtain that the residual vector for the solution $\Delta\hat{\mathbf{h}}(i)$ to the auxiliary system (1) is also equal to $\mathbf{r}(i)$, i.e.,

$$\mathbf{r}(i) = \boldsymbol{\beta}(i) - \mathbf{R}(i)\hat{\mathbf{h}}(i) \tag{9}$$

$$= \boldsymbol{\beta}_0(i) - \mathbf{R}(i)\Delta\hat{\mathbf{h}}(i). \tag{10}$$

Thus, we can now formulate a recursive approach for solving a sequence of systems of equations as presented in Table I. This approach allows us, at each time instant i, instead of solving the original problem (1) with respect to the filter weights $\mathbf{h}(i)$, to deal with an auxiliary problem (6) with respect to the increment of the filter weights $\Delta\mathbf{h}(i)$. The system (6) takes into account the accuracy of the previous solution through the residual vector $\mathbf{r}(i-1)$, as well as the variation of the problem to be solved through the increments $\Delta\mathbf{R}(i)$ and $\Delta\boldsymbol{\beta}(i)$. If a true solution to the system (6) is found then $\hat{\mathbf{h}}(i)$ is the true solution to the problem (1) as well.

When using direct methods for solving the normal equations, both approaches would require approximately the same computational load. However, when using iterative techniques, the new approach is preferable, since it corresponds to solving the original problem with (implicit) initialization by the solution of the

problem for the previous time instant. Therefore, with the same accuracy of calculating the vector $\mathbf{h}(i)$, the proposed approach will typically require a smaller number of iterations. If, in addition to finding a solution vector $\Delta\hat{\mathbf{h}}(i)$, the iterative equation solver produces the residual vector $\mathbf{r}(i)$ at a low computational cost, and a simple way of computing the product $\mathbf{R}(i)\Delta\hat{\mathbf{h}}(i)$ also exists, the complexity of adaptive filtering based on the new approach will be lower than that with the original approach.

Below, the new approach is applied to the exponentially weighted (Section II-A) and sliding window (Section II-B) RLS problems. The new algorithms obtained provide both the filter output and filter weights updated at every time instant i. The most computationally demanding steps of the algorithms are the updating of the correlation matrix $\mathbf{R}(i)$ and the solution of the auxiliary equations. The shifted structure of the input data allows the complexity of the matrix updating to be significantly reduced (Section II-C). Notice that, at step 3, a technique for solving the auxiliary equations should provide both a solution $\Delta\hat{\mathbf{h}}(i)$ and the residual vector $\mathbf{r}(i)$; such techniques are considered in Section III.

A. Exponentially Weighted RLS Algorithm

The exponentially weighted RLS (ERLS) problem deals, at every time instant i, with a N-length data vector $\mathbf{x}(i)$ and a scalar desired signal $z(i)$. An adaptive algorithm should find a vector $\mathbf{h}(i)$ that minimizes the error [2]

$$E_{\mathrm{e}}(i) = \lambda^{i+1}\mathbf{h}^T(i)\mathbf{\Pi}\mathbf{h}(i) + \sum_{j=0}^{i}\lambda^{i-j}[z(j) - \mathbf{h}^T(i)\mathbf{x}(j)]^2 \tag{11}$$

where $\mathbf{\Pi}$ is a regularization matrix and $0 < \lambda \leq 1$ is a forgetting factor. The regularization matrix is usually chosen as a diagonal matrix $\mathbf{\Pi} = \eta\mathbf{I}_N$, where the regularization parameter $\eta > 0$ is a small positive number and $\mathbf{I}_N$ is the $N \times N$ identity matrix [1], [2]. The vector $\mathbf{h}(i)$ can be found by solving the normal equations (1) with the system matrix and the right-hand vector given by [1]

$$\mathbf{R}(i) = \lambda\mathbf{R}(i-1) + \mathbf{x}(i)\mathbf{x}^T(i) \tag{12}$$

$$\boldsymbol{\beta}(i) = \lambda\boldsymbol{\beta}(i-1) + z(i)\mathbf{x}(i). \tag{13}$$

To apply the method in Table I to this problem, the vector $\boldsymbol{\beta}_0(i)$ should be expressed in terms of $\mathbf{x}(i)$ and $z(i)$. From (12) and (13), we obtain

$$\Delta\mathbf{R}(i) = (\lambda-1)\mathbf{R}(i-1) + \mathbf{x}(i)\mathbf{x}^T(i) \tag{14}$$

$$\Delta\boldsymbol{\beta}(i) = (\lambda-1)\boldsymbol{\beta}(i-1) + z(i)\mathbf{x}(i). \tag{15}$$

By using (2) and (14), we obtain

$$\Delta\mathbf{R}(i)\hat{\mathbf{h}}(i-1) = (\lambda-1)[\boldsymbol{\beta}(i-1) - \mathbf{r}(i-1)] + \mathbf{x}(i)y(i) \tag{16}$$

where $y(i)$ is the adaptive filter output at time instant i

$$y(i) = \mathbf{x}^T(i)\hat{\mathbf{h}}(i-1). \tag{17}$$

TABLE II
EXPONENTIALLY WEIGHTED RLS ALGORITHM

Step	Equation	$\times$	$+$
	Initialization: $\hat{\mathbf{h}}(-1) = 0, \mathbf{r}(-1) = 0, \mathbf{R}(-1) = \mathbf{\Pi}$		
	for $i = 0, 1, \ldots$		
1	$\mathbf{R}(i) = \lambda\mathbf{R}(i-1) + \mathbf{x}(i)\mathbf{x}^T(i)$	$2N^2$	N^2
2	$y(i) = \mathbf{x}^T(i)\hat{\mathbf{h}}(i-1)$	N	$N-1$
3	$e(i) = z(i) - y(i)$	$-$	1
4	$\boldsymbol{\beta}_0(i) = \lambda\mathbf{r}(i-1) + e(i)\mathbf{x}(i)$	$2N$	N
5	$\mathbf{R}(i)\Delta\mathbf{h}(i) = \boldsymbol{\beta}_0(i) \Rightarrow \Delta\hat{\mathbf{h}}(i), \mathbf{r}(i)$	P_m	P_a
6	$\hat{\mathbf{h}}(i) = \hat{\mathbf{h}}(i-1) + \Delta\hat{\mathbf{h}}(i)$	$-$	N
	Total: $2N^2 + 3N + P_m$ mult. and $N^2 + 3N + P_a$ adds		

Using (16), we obtain step 2 for the method in Table I:

$$\boldsymbol{\beta}_0(i) = \lambda\mathbf{r}(i-1) + e(i)\mathbf{x}(i) \tag{18}$$

where $e(i)$ is the *a priori* estimation error

$$e(i) = z(i) - y(i). \tag{19}$$

Finally, the exponentially weighted RLS algorithm is summarized in Table II, which also shows the complexity of different steps of the algorithm in terms of multiplications and additions. Notice that the complexity of step 5 depends on the technique used for solving the normal equation. We denote P_m the number of multiplications and P_a the number of additions required by the technique; these figures will be given in Section III.

B. Sliding Window RLS Algorithm

The sliding window RLS (SRLS) problem, at each sample i, deals with finding a vector $\mathbf{h}(i)$ minimizing the error

$$E_{\mathrm{s}}(i) = \mathbf{h}^T(i)\mathbf{\Pi}\mathbf{h}(i) + \sum_{j=i-M+1}^{i}[z(j) - \mathbf{h}^T(i)\mathbf{x}(j)]^2 \tag{20}$$

where M is the sliding window length. Solution to this problem is equivalent to solution of the normal equations (1) where the matrix $\mathbf{R}(i)$ and vector $\boldsymbol{\beta}(i)$ are updated as [2]

$$\mathbf{R}(i) = \mathbf{R}(i-1) + \mathbf{x}(i)\mathbf{x}^T(i) - \mathbf{x}(i-M)\mathbf{x}^T(i-M) \tag{21}$$

$$\boldsymbol{\beta}(i) = \boldsymbol{\beta}(i-1) + z(i)\mathbf{x}(i) - z(i-M)\mathbf{x}(i-M). \tag{22}$$

To find the vector $\boldsymbol{\beta}_0(i)$, we notice that

$$\Delta\mathbf{R}(i)\hat{\mathbf{h}}(i-1) = \mathbf{x}(i)y(i) - \mathbf{x}(i-M)y_M(i) \tag{23}$$

TABLE III
SLIDING WINDOW RLS ALGORITHM

Step	Equation	×	+
	Initialization: for $i<0$: $\mathbf{x}(i)=\mathbf{0}$, $\hat{\mathbf{h}}(-1)=0, \mathbf{r}(-1)=\mathbf{0}, \mathbf{R}(-1)=\mathbf{\Pi}$		
	for $i=0,1,\ldots$		
1	$\mathbf{R}(i)=\mathbf{R}(i-1)+\mathbf{x}(i)\mathbf{x}^T(i)-\mathbf{x}(i-M)\mathbf{x}^T(i-M)$	$2N^2$	$2N^2$
2	$y(i)=\mathbf{x}^T(i)\hat{\mathbf{h}}(i-1)$	N	$N-1$
3	$e(i)=z(i)-y(i)$	–	1
4	$y_M(i)=\mathbf{x}^T(i-M)\hat{\mathbf{h}}(i-1)$	N	$N-1$
5	$e_M(i)=z(i-M)-y_M(i)$	–	1
6	$\boldsymbol{\beta}_0(i)=\mathbf{r}(i-1)+e(i)\mathbf{x}(i)-e_M(i)\mathbf{x}(i-M)$	$2N$	$2N$
7	$\mathbf{R}(i)\Delta\mathbf{h}(i)=\boldsymbol{\beta}_0(i)\Rightarrow\Delta\hat{\mathbf{h}}(i),\mathbf{r}(i)$	P_m	P_a
8	$\hat{\mathbf{h}}(i)=\hat{\mathbf{h}}(i-1)+\Delta\hat{\mathbf{h}}(i)$	–	N
	Total: $2N^2+4N+P_m$ mult. and $2N^2+5N+P_a$ adds		

where $y_M(i)=\mathbf{x}^T(i-M)\hat{\mathbf{h}}(i-1)$, and

$$\Delta\boldsymbol{\beta}(i)=z(i)\mathbf{x}(i)-z(i-M)\mathbf{x}(i-M). \tag{24}$$

From (23) and (24), we obtain step 2 for the method in Table I:

$$\boldsymbol{\beta}_0(i)=\mathbf{r}(i-1)+e(i)\mathbf{x}(i)-e_M(i)\mathbf{x}(i-M) \tag{25}$$

where $e_M(i)=z(i-M)-\mathbf{x}^T(i-M)\hat{\mathbf{h}}(i-1)$. Finally, the sliding window RLS algorithm is summarized in Table III.

C. Transversal RLS Algorithms

The RLS algorithms described in Tables II and III can be used in applications with arbitrary data vectors $\mathbf{x}(i)$, i.e., data vectors with no specific structure. The classical example of such applications is antenna array beamforming [1], [2].

For shift-structured input data

$$\mathbf{x}(i)=[x(i)\;x(i-1)\;\ldots\;x(i-N+1)]^T$$

where $x(i)$ is a discrete-time signal, updating the correlation matrix $\mathbf{R}(i)$ is significantly simplified. The lower-right $(N-1)\times(N-1)$ block of $\mathbf{R}(i)$ can be obtained by copying the upper-left $(N-1)\times(N-1)$ block of $\mathbf{R}(i-1)$. The only part of the matrix $\mathbf{R}(i)$ that should be directly updated is the first row and first column. Due to symmetry of the matrix, it is enough to only calculate the first column. The updating for the exponentially weighted RLS problem is described as

$$\mathbf{R}^{(1)}(i)=\lambda\mathbf{R}^{(1)}(i-1)+x(i)\mathbf{x}(i) \tag{26}$$

TABLE IV
EXPONENTIALLY WEIGHTED TRANSVERSAL RLS ALGORITHM

Step	Equation	×	+
	Initialization: $\hat{\mathbf{h}}(-1)=0$, $\mathbf{r}(-1)=0, \mathbf{R}(-1)=\mathbf{\Pi}$		
	for $i=0,1,\ldots$		
1	$\mathbf{R}^{(1)}(i)=\lambda\mathbf{R}^{(1)}(i-1)+x(i)\mathbf{x}(i)$	$2N$	N
2	$y(i)=\mathbf{x}^T(i)\hat{\mathbf{h}}(i-1)$	N	$N-1$
3	$e(i)=z(i)-y(i)$	–	1
4	$\boldsymbol{\beta}_0(i)=\lambda\mathbf{r}(i-1)+e(i)\mathbf{x}(i)$	$2N$	N
5	$\mathbf{R}(i)\Delta\mathbf{h}(i)=\boldsymbol{\beta}_0(i)\Rightarrow\Delta\hat{\mathbf{h}}(i),\mathbf{r}(i)$	P_m	P_a
6	$\hat{\mathbf{h}}(i)=\hat{\mathbf{h}}(i-1)+\Delta\hat{\mathbf{h}}(i)$	–	N
	Total: $5N+P_m$ mult. and $4N+P_a$ adds		

TABLE V
SLIDING WINDOW TRANSVERSAL RLS ALGORITHM

Step	Equation	×	+
	Initialization: for $i<0$: $x(i)=0$ $\hat{\mathbf{h}}(-1)=0, \mathbf{r}(-1)=\mathbf{0}, \mathbf{R}(-1)=\mathbf{\Pi}$		
	for $i=0,1,\ldots$		
1	$\mathbf{R}^{(1)}(i)=\mathbf{R}^{(1)}(i-1)+x(i)\mathbf{x}(i)-x(i-M)\mathbf{x}(i-M)$	$2N$	$2N$
2	$y(i)=\mathbf{x}^T(i)\hat{\mathbf{h}}(i-1)$	N	$N-1$
3	$e(i)=z(i)-y(i)$	–	1
4	$y_M(i)=\mathbf{x}^T(i-M)\hat{\mathbf{h}}(i-1)$	N	$N-1$
5	$e_M(i)=z(i-M)-y_M(i)$	–	1
6	$\boldsymbol{\beta}_0(i)=\mathbf{r}(i-1)+e(i)\mathbf{x}(i)-e_M(i)\mathbf{x}(i-M)$	$2N$	$2N$
7	$\mathbf{R}(i)\Delta\mathbf{h}(i)=\boldsymbol{\beta}_0(i)\Rightarrow\Delta\hat{\mathbf{h}}(i),\mathbf{r}(i)$	P_m	P_a
8	$\hat{\mathbf{h}}(i)=\hat{\mathbf{h}}(i-1)+\Delta\hat{\mathbf{h}}(i)$	–	N
	Total: $6N+P_m$ mult. and $7N+P_a$ adds		

and, for the sliding window RLS problem

$$\mathbf{R}^{(1)}(i)=\mathbf{R}^{(1)}(i-1)+x(i)\mathbf{x}(i)-x(i-M)\mathbf{x}(i-M). \tag{27}$$

The transversal ERLS and SRLS algorithms are presented in Tables IV and V, respectively.

TABLE VI
EXACT LINE SEARCH METHOD

Step	Equation
	Initialization: $\Delta\hat{\mathbf{h}} = \mathbf{0}, \mathbf{r} = \boldsymbol{\beta}_0$
	for $k = 1, \ldots, N_u$
1	Choose $\mathbf{d}$ such that $\mathbf{d}^T\mathbf{r} \neq \mathbf{0}$
2	$\mathbf{v} = \mathbf{R}\mathbf{d}$
3	$\alpha = \mathbf{d}^T\mathbf{r}/\mathbf{d}^T\mathbf{v}$
4	$\Delta\hat{\mathbf{h}} = \Delta\hat{\mathbf{h}} + \alpha\mathbf{d}$
5	$\mathbf{r} = \mathbf{r} - \alpha\mathbf{v}$

III. LINE SEARCH METHODS

Many techniques can be used for solving the auxiliary normal equations (6). We are interested here in iterative algorithms as opposed to the direct solution algorithms because of lower computational complexity of the former. Specifically, we will consider line search methods that provide both a solution vector $\Delta\hat{\mathbf{h}}$ and the residual vector $\mathbf{r}$, which are required for applying the approach described in Table I. In this section, for clarity we omit the time index i from matrix and vector notations.

Solving the normal equations (6) is equivalent to minimizing the quadratic function

$$f(\Delta\mathbf{h}) = \frac{1}{2}\Delta\mathbf{h}^T\mathbf{R}\Delta\mathbf{h} - \Delta\mathbf{h}^T\boldsymbol{\beta}_0. \tag{28}$$

In a line search method, at each iteration k, the solution $\Delta\hat{\mathbf{h}}$ is updated in a direction $\mathbf{d}$ that is chosen to be non-orthogonal to the residual vector $\mathbf{r}$, i.e., $\mathbf{d}^T\mathbf{r} \neq 0$. The step size α minimizing the function $f(\Delta\hat{\mathbf{h}} + \alpha\mathbf{d})$ is $\alpha = \mathbf{d}^T\mathbf{r}/\mathbf{d}^T\mathbf{R}\mathbf{d}$; this step size α corresponds to the exact line search method [14]–[16]. A general description of the exact line search method [13] is given in Table VI, where N_u denotes the number of iterations.

The conjugate gradient (CG) [13] and coordinate descent (CD) algorithms considered below in Sections III-A and -B, respectively, are examples of the exact line search method. Inexact line search methods, though not providing the maximum decrement for a particular iteration, can improve the convergence speed in a sequence of iterations [15], [17]. The dichotomous coordinate descent (DCD) algorithm presented in Section III-C, is an inexact line search method.

A. Conjugate Gradient Algorithm

An efficient variant of the line search method is the CG algorithm [13] shown in Table VII. At the first iteration, $k = 1$, the direction vector is the residual vector: $\mathbf{d} = \mathbf{r}$. At other iterations, $k > 1$, the direction $\mathbf{d}$ is updated to guarantee $\mathbf{R}$-conjugacy of the direction vectors. Due to its fast convergence, the CG method has already been used for adaptive filtering for a long time (e.g., see [5], [7], [18], [8] and references therein). Although, the CG algorithm shows fast convergence (as will be seen from simulation results in Section V), its complexity is too high for fast adaptive filtering. In general, the complexity of the algorithm is $\mathcal{O}(N^2)$ per update. The algorithm also requires divisions at steps 1 and 3.

TABLE VII
CONJUGATE GRADIENT ALGORITHM

Step	Equation	$\times$	$+$
	Initialization: $\Delta\hat{\mathbf{h}} = \mathbf{0}, \mathbf{r} = \boldsymbol{\beta}_0$, $\rho_0 = \mathbf{r}^T\mathbf{r}, \mathbf{d} = \mathbf{r}$		
	for $k = 1, \ldots, N_u$		
1	if $k > 1, \mathbf{d} = \mathbf{r} + (\rho_{k-1}/\rho_{k-2})\mathbf{d}$	N	N
2	$\mathbf{v} = \mathbf{R}\mathbf{d}$	N^2	N^2-N
3	$\alpha = \rho_{k-1}/\mathbf{d}^T\mathbf{v}$	N	$N-1$
4	$\Delta\hat{\mathbf{h}} = \Delta\hat{\mathbf{h}} + \alpha\mathbf{d}$	N	N
5	$\mathbf{r} = \mathbf{r} - \alpha\mathbf{v}$	N	N
6	$\rho_k = \mathbf{r}^T\mathbf{r}$	N	$N-1$
	Total: $P_m = (N^2+5N)N_u$ and $P_a = (N^2+4N-2)N_u$		

It can be shown that the CG adaptive algorithm as described in Tables V and VII for the particular case $N_u = M$ produces the same filter weights and output signal as the affine projection algorithm [2] of projection order M and the CG adaptive algorithm proposed in [5].

B. Coordinate Descent (CD) Algorithm

If the directions are chosen as Euclidean coordinates, i.e., $\mathbf{d} = \mathbf{e}_p$, where all elements of the vector $\mathbf{e}_p$ are zeros, except the pth element that is equal to one, the iterations are significantly simplified. In this case, for the exact line search, $\mathbf{v} = \mathbf{R}\mathbf{d} = \mathbf{R}^{(p)}$ is the pth column of the correlation matrix $\mathbf{R}$. Thus, the most complicated step of the line search method (step 2 in Table VI), requiring the matrix-vector multiplication of complexity $\mathcal{O}(N^2)$, is completely eliminated. Moreover, the other steps are also simplified $\mathbf{d}^T\mathbf{r} = r_p, \mathbf{d}^T\mathbf{v} = R_{p,p}, \alpha = r_p/R_{p,p}$, and $\Delta\hat{h}_p = \Delta\hat{h}_p + \alpha$. If the directions are chosen in a cyclic order $p = 1, \ldots, N$, we arrive at Gauss-Seidel iterations, and the EDS algorithm of complexity $\mathcal{O}(N^2)$ [11]. However, such choice is not efficient in our case, as it requires at least N iterations at a time instant, resulting in high complexity. Attempts to distribute the N Euclidean directions in time by assigning one direction to one time instant has led to the fast EDS algorithm [11], [9]. The maximum complexity of the fast EDS algorithm (including the filtering) is $(8N + 5)$ multiplications per time instant [11]. However, the convergence of the fast EDS algorithms is slow [11], [19]. Moreover, our simulation results (not presented here) show that the fast EDS algorithm is sensitive to the order of updating the filter weights and experiences instability at the initial part of the learning process. A more efficient method for selecting the leading index p is therefore important to speed up the convergence.

For the exact line search method in Table VI, we have [13]

$$f(\Delta\mathbf{h} + \alpha\mathbf{d}) = f(\Delta\mathbf{h}) - \frac{1}{2}\frac{(\mathbf{d}^T\mathbf{r})^2}{\mathbf{d}^T\mathbf{R}\mathbf{d}}. \tag{29}$$

TABLE VIII
COORDINATE DESCENT (CD) ALGORITHM

Step	Equation	×	+
	Initialization: $\Delta\hat{\mathbf{h}} = \mathbf{0}, \mathbf{r} = \boldsymbol{\beta}_0$		
	for $k = 1, \ldots, N_u$		
1	$p = \arg\max_{n=1,\ldots,N}\{\lvert r_n\rvert\}$	$-$	$N-1$
2	$\alpha = r_p / R_{p,p}$	$-$	$-$
3	$\Delta\hat{h}_p = \Delta\hat{h}_p + \alpha$	$-$	1
4	$\mathbf{r} = \mathbf{r} - \alpha\mathbf{R}^{(p)}$	N	N
	Total: $P_m = NN_u$ and $P_a = 2NN_u$		

The (nonnegative) term $(\mathbf{d}^T\mathbf{r})^2(\mathbf{d}^T\mathbf{R}\mathbf{d})^{-1}$ shows how quickly the function $f(\cdot)$ decreases at an update. For an exact coordinate search, we have

$$f(\Delta\mathbf{h} + \alpha\mathbf{e}_p) = f(\Delta\mathbf{h}) - \frac{1}{2}\frac{r_p^2}{R_{p,p}}. \tag{30}$$

If the matrix $\mathbf{R}$ is calculated by averaging over a relatively long time interval, $R_{p,p}$ are approximately constant over p. Therefore, the coordinate direction $\mathbf{e}_p$ chosen according to

$$p = \arg\max_{n=1,\ldots,N}\{\lvert r_n\rvert\} \tag{31}$$

at a particular iteration, will provide the largest decrement of $f(\cdot)$. The CD algorithm with the leading index (31) is presented in Table VIII. One update in the algorithm requires only N multiplications and $2N$ additions. Note that the CD algorithm is also known as Southwell's relaxation method [20], [17]. Its convergence to the optimal solution for the normal equations follows from the following.

Theorem [17]: If the leading index p of a relaxation coordinate descent process

$$\Delta\hat{\mathbf{h}}_k = \Delta\hat{\mathbf{h}}_{k-1} + q_k\frac{r_p}{R_{p,p}}\mathbf{e}_p$$

for a linear system of equations with a positive-definite matrix $\mathbf{R}$ is chosen such that, at each iteration k

$$\lvert r_p\rvert \geq \gamma\lvert r_n\rvert, \quad (0 < \gamma \leq 1, \quad n = 1, \ldots, N)$$

and if $\varepsilon < q_k < 2-\varepsilon, 0 < \varepsilon < 1$, then the process converges to the optimal solution $\Delta\mathbf{h}$ of the system and there exists a number $\theta, 0 < \theta < 1$, such that $\lvert\Delta\mathbf{h} - \Delta\hat{\mathbf{h}}_k\rvert < \theta^k$.

In our case, for all k, we have $\gamma = 1$ and $q_k = 1$. Therefore, the theorem can be directly applied to the CD algorithm in Table VIII.

It can be shown that the RLS algorithm based on the CD iterations with the leading index (31) produces the same filter weights and output signal as that of the recursive adaptive matching pursuit (RAMP) algorithm proposed in [21].

TABLE IX
DCD ALGORITHM

Step	Equation	+
	Initialization: $\Delta\hat{\mathbf{h}} = 0, \mathbf{r} = \boldsymbol{\beta}_0, \alpha = H, q = 0$	
	for $m = 1, \ldots, M_b$	
1	$\alpha = \alpha/2$	
2	flag $= 0$	
	for $n = 1, \ldots, N$	
3	if $\lvert r_n\rvert > (\alpha/2)R_{n,n}$ then	1
4	$\Delta\hat{h}_n = \Delta\hat{h}_n + \text{sign}(r_n)\alpha$	1
5	$\mathbf{r} = \mathbf{r} - \text{sign}(r_n)\alpha\mathbf{R}^{(n)}$	N
6	$q = q + 1$, flag $= 1$	$-$
7	if $q > N_u$, the algorithm stops	$-$
8	if flag $= 1$, then repeat step 2	$-$
	Total: $P_m = 0$ and $P_a \leq N(2N_u + M_b - 1) + N_u$	

C. Dichotomous Coordinate Descent (DCD) Algorithm

The DCD algorithm [22] is presented in Table IX. It updates the solution in directions of Euclidean coordinates in the cyclic order $(n = 1, \ldots, N)$. Such choice of directions is used in the EDS algorithm. However, in the DCD algorithm, the step-size α is chosen in a different way—it takes on one of M_b predefined values corresponding to binary representation of elements of $\Delta\hat{\mathbf{h}}$ with M_b bits within an amplitude range $[-H, H]$. The algorithm starts the iterative search from the most significant bits of elements in $\Delta\hat{\mathbf{h}}$. As the most significant bits have been updated, the algorithm starts updating the next less significant bit, and so on. Due to the quantized step-size, there are 'unsuccessful' iterations (decided at step 3) without updates of the solution and "successful" iterations where the solution and the residual vector are updated (steps 4 and 5). The DCD algorithm as described above can be found in [23].

The complexity of the DCD algorithm depends on the implementation platform. In [23], it is estimated for software implementation, which usually requires an extra operation for calculating $\lvert r_n\rvert$ at step 3. For a hardware implementation, in which we are interested here, step 3 can be considered as one addition since calculation of $\lvert r_n\rvert$ can be incorporated in an adder used for the comparison. The complexity can be considered as a random number with an upper bound corresponding to a worst-case scenario as follows. For an mth bit, $m = 1, \ldots, M_b - 1$, within one pass $(n = 1, \ldots, N)$ there is one "successful" iteration and then, in another pass, N "unsuccessful" iterations; this will require $(3N+1)(M_b-1)$ additions. For the last (least significant) bit, $m = M_b$, there are (N_u-M_b+1) passes each with one "successful" iteration; this will require $(N_u-M_b+1)(2N+1)$ additions. Thus, the worst-case complexity is $N(2N_u+M_b-1)+N_u$ additions. Notice that the average complexity will be lower.

TABLE X
DCD ALGORITHM WITH LEADING ELEMENT

Step	Equation	+
	Initialization: $\Delta\hat{\mathbf{h}}=0, \mathbf{r}=\boldsymbol{\beta}_0, \alpha=H/2, m=1$	
	for $k=1,\ldots,N_u$	
1	$p=\arg\max_{n=1,\ldots,N}\{\lvert r_n\rvert\}$, go to step 4	$N-1$
2	$m=m+1, \alpha=\alpha/2$	–
3	if $m>M_b$, the algorithm stops	–
4	if $\lvert r_p\rvert \le (\alpha/2)R_{p,p}$, then go to step 2	1
5	$\Delta\hat{h}_p=\Delta\hat{h}_p+\text{sign}(r_p)\alpha$	1
6	$\mathbf{r}=\mathbf{r}-\text{sign}(r_p)\alpha\mathbf{R}^{(p)}$	N
	Total: $P_m=0$ and $P_a \le (2N+1)N_u+M_b$	

However, in hardware implementation one should take into account the worst-case complexity.

If $N_u \gg M_b$, the complexity of the DCD algorithm in Table IX is approximately upper bounded by $2NN_u$. However, if the number of updates N_u is small (which is the case that we are interested in here), the term NM_b will dominate in the DCD complexity. A computationally more efficient variant of the DCD algorithm can be proposed that eliminates this term. This new version of the DCD algorithm finds a 'leading' (pth) element in $\Delta\hat{\mathbf{h}}$ to be updated similarly to the CD algorithm in Table VIII. The new DCD algorithm is shown in Table X. It is seen that one update in the DCD algorithm requires N bit-shifts, N additions, and N comparisons; the latter can be counted as additions. With N_u updates, the complexity of the DCD algorithm is upper limited by $(2N+1)N_u+M_b$ additions. This corresponds to a worst-case scenario when the algorithm in Table X performs all N_u updates, i.e., the condition at step 3 is never satisfied. The important property of the DCD algorithm is that it requires no multiplication, no division, and no square root operations. Note also that the parameter N_u defines a maximum number of filter weights that can be updated at a time instant. Thus, adaptive filtering based on coordinate descent search and, in particular, on the DCD algorithm, implements a *selective partial update* [24].

It is seen from the algorithm description that, in any iteration k at step 4, the step size α satisfies the relationship $|r_p|/R_{p,p} \le \alpha < 2|r_p|/R_{p,p}$ which results in $1 \le q_k < 2$. Strictly speaking, this means that conditions of the convergence theorem in Section III-B are not satisfied at every iteration ($\varepsilon=0$). However, as step 4 in Table X describes a quantization process, we can assume that the parameter q_k is uniformly distributed on [1, 2) and, therefore, with probability 1, these conditions will be satisfied. Note that the decrement of the cost function (28) at every iteration is given by

$$f(\Delta\hat{\mathbf{h}})-f\left(\Delta\hat{\mathbf{h}}+q_k\frac{r_p}{R_{p,p}}\right)=\frac{r_p^2}{R_{p,p}}q_k\left(1-\frac{q_k}{2}\right) \quad (32)$$

which shows that the cost function decreases at every iteration.

TABLE XI
COMPLEXITY OF PROPOSED AND KNOWN TRANSVERSAL ADAPTIVE ALGORITHMS

Algorithm	×	+	÷
ERLS-CG	$N^2N_u+5NN_u+3N$	$N^2N_u+4NN_u+6N$	$2N_u-1$
ERLS-CD	NN_u+3N	$2NN_u+6N$	N_u
ERLS-DCD	$3N$	$2NN_u+6N$	–
SRLS-CG	$N^2N_u+5NN_u+6N$	$N^2N_u+4NN_u+7N$	$2N_u-1$
SRLS-CD	NN_u+6N	$2NN_u+7N$	N_u
SRLS-DCD	$6N$	$2NN_u+7N$	–
RLS	N^2+5N+1	N^2+3N	1
NLMS	$2N+3$	$2N+3$	1

If, in the DCD algorithm described in Table X, step 4 is replaced with $|r_p| \le (\alpha/4)R_{p,p}$, we obtain $|r_p|/2R_{p,p} \le \alpha < |r_p|/R_{p,p}$ which results in $1/2 \le q_k < 1$. In this case, the convergence theorem is satisfied for $0 < \varepsilon \le 1/2$. However, this choice slows down the convergence. Multiple experiments have shown that the DCD algorithm with $1 \le q_k < 2$, presented in Table X, is preferable over that with $1/2 \le q_k < 1$.

IV. PRACTICAL ISSUES

In this section, we address some practical issues related to implementation of the proposed adaptive algorithms.

For the exponentially weighted RLS algorithm in Table II, the matrix update, in the general case, requires $2N^2$ multiplications and N^2 additions. However, if the forgetting factor is chosen as $\lambda=1-2^{-s}$ with a positive integer s, then the multiplication by λ in step 1 can be replaced by addition and bit-shift operations, thus giving the total number of multiplications and additions N^2 and $2N^2$, respectively. Similar approach is also applicable to the exponentially weighted transversal RLS algorithm in Table IV, thus reducing the number of multiplications to N and increasing the number of additions to $2N$. Moreover, calculation of $\boldsymbol{\beta}_0(i)$ at step 4 is also simplified to N multiplications and $2N$ additions. However, even if λ is chosen differently, it is not difficult to accurately approximate it by a number making the multiplication by λ simple for implementation.

In the transversal adaptive filters, the direct copying of the $(N-1)\times(N-1)$ matrix block would require significant processor time. To avoid the copying, a simple memory address modification can be performed, when the block does not change its position in the memory and only the row and column addresses are updated. This address update was used in our FPGA design described here.

Table XI shows the complexity of the proposed and known transversal adaptive filters; the complexity for the RLS and NLMS algorithms is from [2]. The complexity for the ERLS algorithms takes into account the choice of the forgetting factor as $\lambda=1-2^{-s}$ with a positive integer s. For additions, we only show figures that are $\mathcal{O}(N^2)$ or $\mathcal{O}(N)$ and ignore figures that are $\mathcal{O}(N_u), \mathcal{O}(M_b)$ or $\mathcal{O}(1)$. It is seen that the transversal

TABLE XII
FPGA RESOURCES FOR ERLS-DCD ALGORITHMS

Algorithms:	ERLS-DCD		Transversal ERLS-DCD	
Resources	$N = 16$	$N = 64$	$N = 16$	$N = 64$
Slice	1154 (8.4%)	1225 (8.9%)	1222 (8.9%)	1398 (10%)
D Flip-Flop	613 (2.2%)	669 (2.4%)	615 (2.2%)	680 (2.5%)
LUT-4	2112 (7.7%)	2247 (8.2%)	2256 (8.2%)	2496 (9.1%)
Block RAM	4 (2.9%)	11 (8.1%)	4 (2.9%)	11 (8.1%)
Multiplier	3 (2.2%)	3 (2.2%)	3 (2.2%)	3 (2.2%)
Update rate	203 kHz	31 kHz	256 kHz	80 kHz

ERLS-DCD algorithm requires only $3N$ multiplications per sample and no division. The transversal SRLS-DCD algorithm requires $6N$ multiplications and no division.

The ERLS-DCD algorithms have been implemented on a Xilinx Virtex-II Pro Development System with a XC2VP30 FPGA running at 100 MHz.[1] VHDL was used to describe the design, and the Xilinx ISE 8.1 was used for synthesizing and downloading the design to the target platform.[2] The input data $\mathbf{x}(i)$ are represented in 16-bit fixed-point Q15 format [25]. The desired signal $z(i)$ is represented by 32 bits in the Q15 format. The matrix $\mathbf{R}(i)$ and vectors $\mathbf{r}(i), \boldsymbol{\beta}_0(i)$, and $\hat{\mathbf{h}}(i)$ are represented by 32 bits in the Q15 format. When computing the filter output $y(i)$, each multiplication results in 47 bits in the Q30 format; after accumulation, $y(i)$ is truncated to 32 bits in the Q15 format. The error signal $e(i)$ is then represented by 32 bits in the Q15 format. The forgetting factor is chosen as $\lambda = 1 - 2^{-s}$, where s is an integer; thus, the multiplications $\lambda\mathbf{R}(i)$ and $\lambda\mathbf{r}(i)$ are replaced by bit-shifts and additions.

The FPGA resources for four designs are presented in Table XII. Two figures are shown for every resource: number of elements used and the percentage of the resource available on the FPGA device. The ERLS-DCD algorithm as described in Tables II and X was implemented for the cases $N = 16$ and $N = 64$ with $N_u = 16$. This design is suitable for arbitrary data vectors $\mathbf{x}(i)$, e.g., it is applicable for adaptive antenna beamforming. For an 8-element antenna we obtain the update rate 205 kHz which is approximately 60 times higher than that of a design based on the QRD-RLS algorithm for 9-element antenna and with approximately the same chip area [26], [27]. The transversal ERLS-DCD algorithm with $N_u = 16$ is implemented by using a serial design of the DCD algorithm [28] with $N + 3$ (100 MHz) cycles for one update. The update rate can be increased by reducing N_u and/or using a parallel design of the DCD algorithm [27]. It is seen that the whole design requires at most 10% of the resources available on the FPGA device. More details on FPGA implementation of the proposed adaptive filtering algorithms will be presented in a separate paper. We have carried out many numerical experiments with these designs, in particular, a long-time experiment where 10^7 vectors $\mathbf{x}(i)$ were processed. No instability problem was observed during the experiments.

[1][Online] Available: http://www.xilinx.com

[2]See footnote 1.

V. NUMERICAL RESULTS

Here, we present results obtained by computer simulation. We compare the mean squared error (MSE) performance of the proposed adaptive algorithms against the classical exponentially weighted RLS algorithm, NLMS algorithm, and a recently proposed efficient conjugate gradient control Liapunov function (CG-CLF) algorithm with complexity $\mathcal{O}(N^2)$ [8]. Only scenarios with the time-shifted structure of input data, corresponding to the transversal adaptive filter, are considered. The input data are generated according to

$$z(i) = \mathbf{h}^T(i)\mathbf{x}(i) + n(i) \tag{33}$$

where $n(i)$ is the additive zero-mean Gaussian random noise with variance σ^2. The vector $\mathbf{x}(i) = [x(i)\ x(i-1)\ \ldots\ x(i-N+1)]^T$ contains either a real speech signal or autoregressive correlated random numbers given by

$$x(i) = \nu x(i-1) + w(i) \tag{34}$$

where ν is the autoregressive factor $(0 \leq \nu < 1)$ and $w(i)$ are uncorrelated zero-mean random Gaussian numbers of unit variance. The MSE in a simulation trial is calculated as

$$\varepsilon(i) = \frac{[\mathbf{h}(i) - \hat{\mathbf{h}}(i)]^T[\mathbf{h}(i) - \hat{\mathbf{h}}(i)]}{\mathbf{h}^T(i)\mathbf{h}(i)}. \tag{35}$$

The MSEs obtained in N_{mc} trials are averaged and plotted against the time index i. Results in Figs. 1 to 5 below are obtained by floating point simulation. Fig. 6 compares floating and fixed point simulation results.

Fig. 1 shows the MSE performance of the ERLS-CG and ERLS-DCD algorithms against the RLS, NLMS, and CG-CLF algorithms. All elements of the impulse response $\mathbf{h}(i)$ are kept constant over the first 1000 samples; the elements are independent random numbers uniformly distributed on $[-1, +1]$. At time instant $i = 1000$, a new vector $\mathbf{h}$ is generated and kept constant over the remaining samples. It is seen that, in the case of $N_u = 1$, the ERLS-DCD algorithm outperforms the ERLS-CG algorithm, but is inferior to the CG-CLF algorithm. For $N_u = 2$, the ERLS-DCD and CG-CLF algorithms demonstrate similar performance, whereas the ERLS-CG algorithm converges faster. For $N_u \geq 4$, the ERLS-DCD and ERLS-CG algorithms outperform the CG-CLF algorithm. For a fixed N_u, the ERLS-CG algorithm converges faster than the ERLS-DCD algorithm. However, this is achieved at the expense of a significant increase in the complexity (see Table XIII). Under a fixed complexity, the ERLS-DCD algorithm provides significantly faster convergence than the ERLS-CG algorithm. Fig. 1(b) shows that after a change of the impulse response, only two updates $(N_u = 2)$ are enough for both the ERLS-CG and ERLS-DCD algorithms to approach the RLS performance. The results for $N_u > 2$ are not shown as they are not distinguishable from that of the classical RLS algorithm.

Fig. 2 compares the performance of the ERLS-CD and ERLS-DCD algorithms. It is seen that, with increase in N_u, the ERLS-CD algorithm approaches the RLS performance.

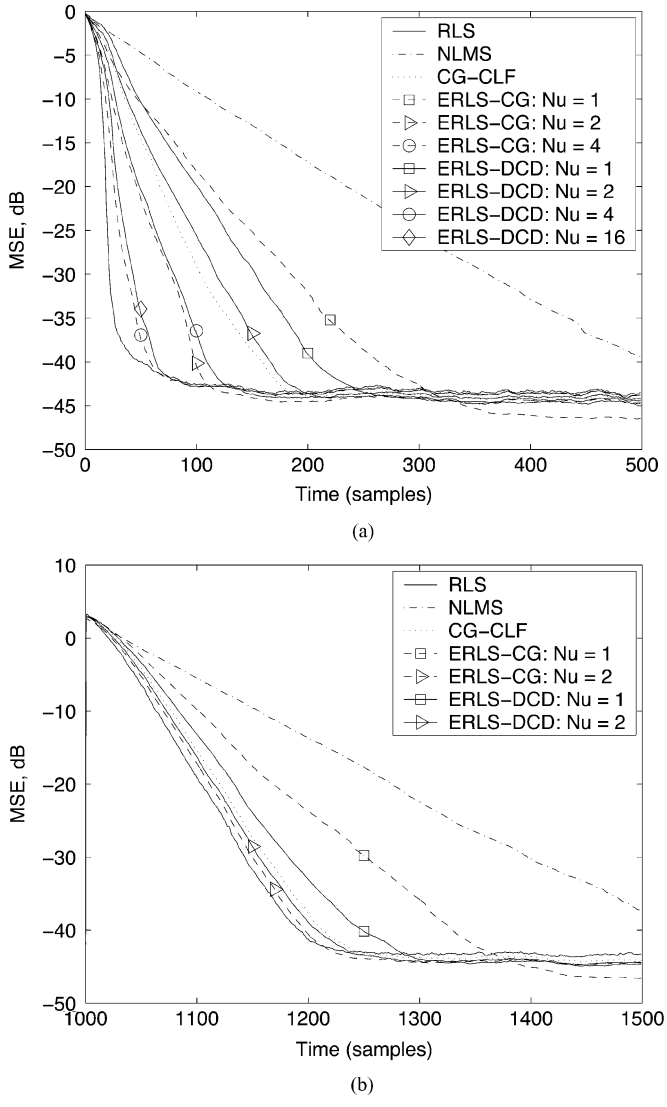


Fig. 1. MSE performance of the ERLS-CG and ERLS-DCD algorithms against the RLS, NLMS, and CG-CLF algorithms; $N = 16, \lambda = 1 - 1/(2N) \approx 0.969, \eta = 10^{-3}, H = 1, M_b = 16, \nu = 0.9, \sigma = 0.01, N_{mc} = 100$. (a) Initial convergence. (b) Convergence after a change of the impulse response.

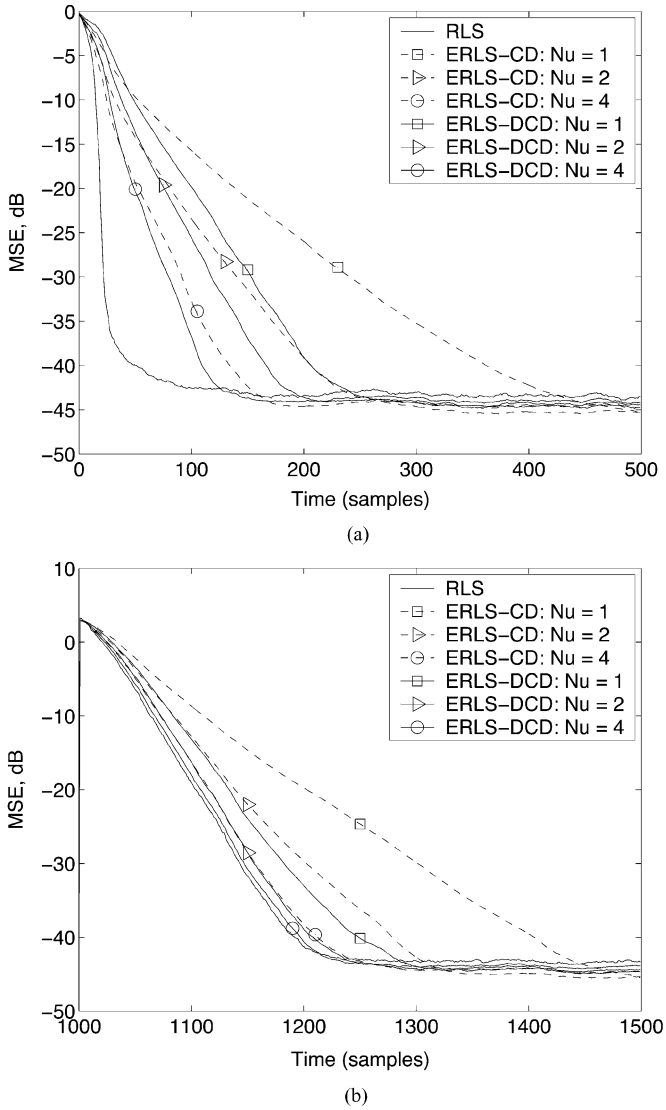


Fig. 2. MSE performance of the ERLS-CD and ERLS-DCD algorithms against the RLS and NLMS algorithms; $N = 16, \lambda = 1 - 1/(2N) \approx 0.969, \eta = 10^{-3}, H = 1, M_b = 16, \nu = 0.9, \sigma = 0.01, N_{mc} = 100$. (a) Initial convergence. (b) Convergence after a change of the impulse response.

However, the performance of the ERLS-DCD algorithm is superior to that of the ERLS-CD and, as seen from Table XIII, it requires a significantly fewer number of multiplications.

Fig. 3 shows the MSE performance of the SRLS-CG and SRLS-DCD algorithms against the RLS and NLMS algorithms. Although the SRLS-DCD algorithm has a slightly higher complexity than the ERLS-DCD algorithm, it achieves the same steady-state MSE more quickly, after a change of the impulse response. Therefore, for some applications, it will be beneficial to use the SRLS-DCD algorithm. In similarity to results for the ERLS algorithms, the SRLS-DCD requires more updates than the SRLS-CG algorithm to achieve the same convergence speed. However, the SRLS-DCD algorithm has significantly lower complexity.

The results in Fig. 4 correspond to the application of adaptive filtering to acoustic echo cancellation with a long impulse response, $N = 512$. Elements of the impulse response $h_n, n = 0, \ldots, N-1$, are independent zero-mean random numbers with variance $\exp(-0.005n)$, which corresponds to a typical acoustic impulse response [29]. The vectors $\mathbf{x}(i)$ contain a real speech signal sampled at a frequency of 8 kHz. It is seen that with $N_u = 1$, the ERLS-DCD algorithm significantly outperforms the NLMS algorithm. With increase in N_u, the MSE performance of the ERLS-DCD algorithm is significantly improved and, in the steady state, for $N_u \geq 2$, it outperforms the RLS algorithm. Table XIV shows the complexity of the three algorithms. It is seen that the complexity of the ERLS-DCD algorithm is significantly lower than that of the RLS algorithm and it requires only 50% more multiplications than the NLMS algorithm.

Fig. 5 shows the tracking performance of the ERLS-DCD algorithm in a time-varying environment. The nth element $h_n(i)$ of the impulse response $\mathbf{h}(i)$ varies in time according to

$$h_n(i) = h_n(0)\cos(2\pi F i + \phi_n) \tag{36}$$

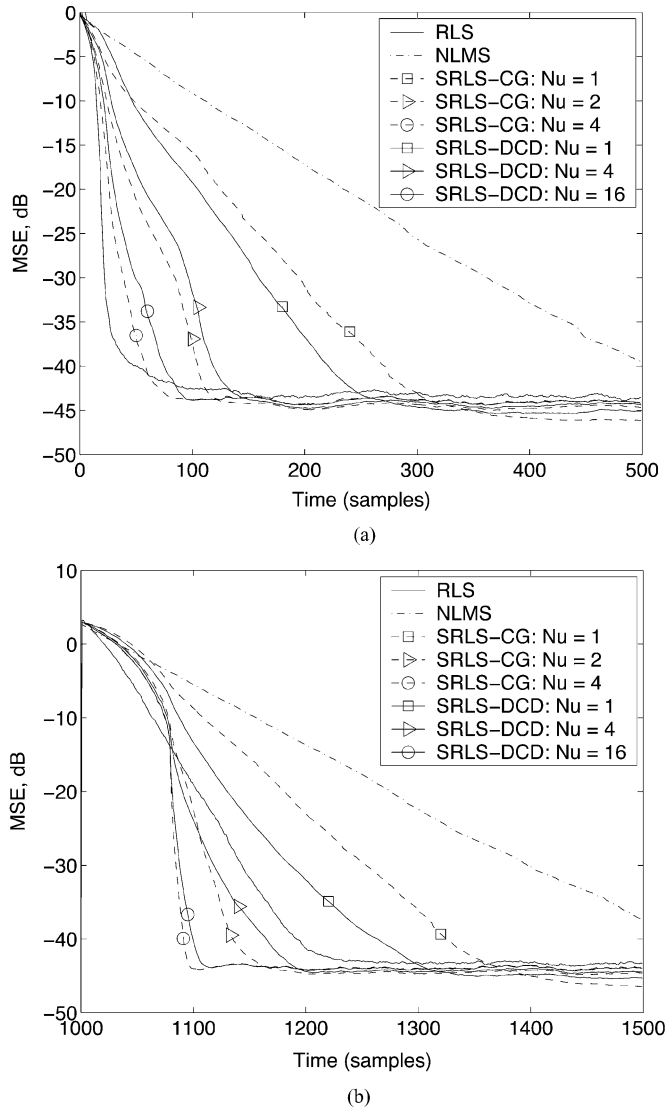


Fig. 3. MSE performance of the SRLS-CG and SRLS-DCD algorithms against the RLS and NLMS algorithms; $N = 16, \lambda = 1-1/(2N) \approx 0.969$ (for RLS), $\eta = 10^{-3}, M = 5N, H = 1, M_b = 16, \nu = 0.9, \sigma = 0.01, N_{mc} = 100$. (a) Initial convergence. (b) Convergence after a change of the impulse response.

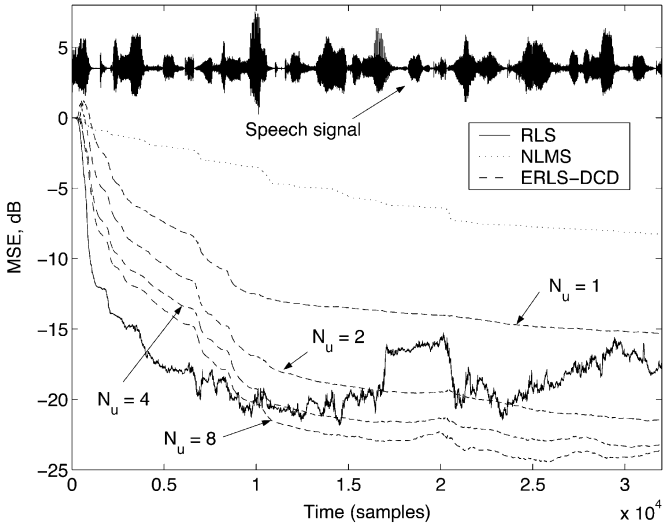


Fig. 4. Echo cancellation experiment with a real speech signal. MSE performance of the ERLS-DCD versus RLS and NLMS algorithms: $N = 512$, SNR = 30 dB, $\lambda = 1-1/(4N) \approx 0.9995, \eta = 0.015, H = 1, M_b = 16, N_{mc} = 1$.

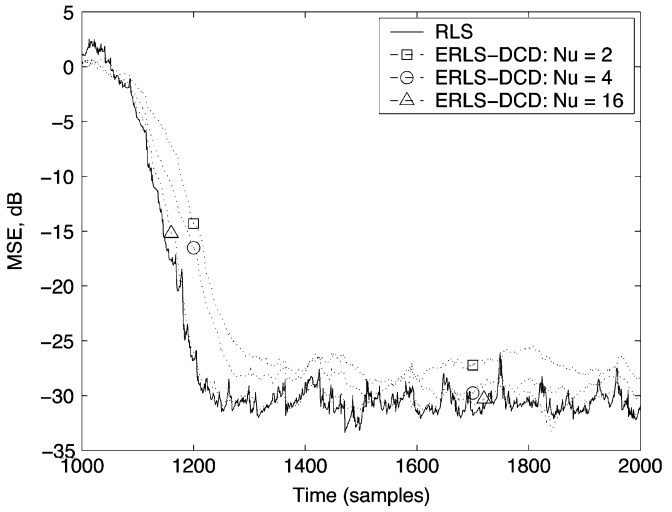


Fig. 5. The tracking performance of the ERLS-DCD algorithm in a time-varying environment: $F = 10^{-4}, \nu = 0.9, \sigma = 0.001, N = 64, \lambda = 0.975, \eta = 10^{-3}, N_{mc} = 1$.

where ϕ_n are independent random numbers uniformly distributed on $[-\pi, \pi)$, $h_n(0)$ are independent zero-mean Gaussian random numbers of unit variance, and F is the variation rate. It is seen that as N_u increases, the MSE performance of the ERLS-DCD algorithm is approaching that of the RLS algorithm.

Fig. 6 shows the performance of a fixed-point implementation of the ERLS-DCD algorithm against the ERLS-DCD and classical RLS algorithms implemented in floating point. For representation of all variables in the algorithm, including the input data $z(i)$ and $\mathbf{x}(i)$, elements of the matrix $\mathbf{R}$ and vector $\mathbf{r}$, etc., N_b bits are used ($N_b = 16$ or $N_b = 24$). It can be seen that the accuracy of both the fixed-point ERLS-DCD and floating-point ERLS-DCD algorithms depends on the parameter M_b that defines the number of bits for representation of the solution vector $\hat{\mathbf{h}}$. As M_b increases, the steady-state MSE approaches that of the RLS algorithm. For the fixed-point ERLS-DCD algorithm, for a fixed M_b, the steady-state MSE depends on N_b. In this scenario, for $M_b = 16$, the parameter $N_b = 16$ limits the algorithm performance, while $N_b = 24$ provides enough accuracy to achieve the floating-point performance.

VI. Conclusion

In this paper, we have derived low-complexity RLS adaptive filtering algorithms. The RLS problem is represented as a sequence of auxiliary normal equations which are then approximately solved by using iterative line search methods. A new variant of the DCD algorithm is proposed; the use of the DCD algorithm as a line search method has led to implementation of the exponentially weighted and sliding window transversal RLS algorithms by using only $3N$ and $6N$ multiplications per sample, respectively. Simulation results show that the performance of the proposed adaptive algorithms can be made arbitrarily close to that of the classical RLS algorithm. The convergence properties of the proposed algorithms were discussed. A

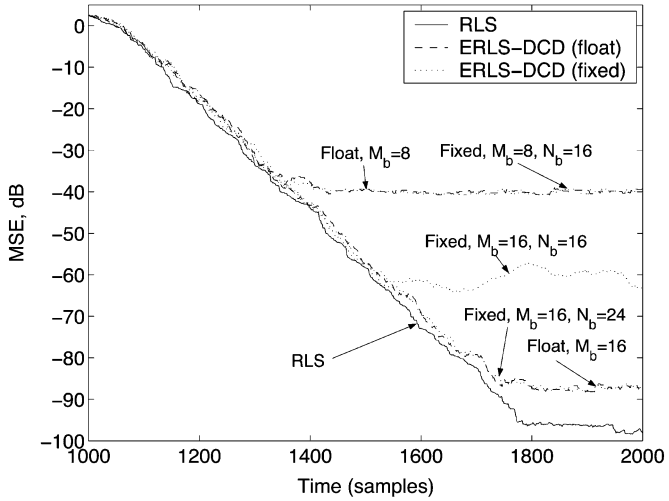


Fig. 6. The MSE performance of a fixed-point implementation of the ERLS-DCD algorithm against the floating point ERLS-DCD and classical RLS algorithms: $N = 64, \sigma = 10^{-5}, \lambda = 1 - 1/N \approx 0.984, \eta = 2^{-10} \approx 10^{-3}, N_u = 2, \nu = 0, N_{mc} = 1$.

TABLE XIII
COMPLEXITY OF ADAPTIVE ALGORITHMS ($N = 16$)

Algorithm	×	+	÷
ERLS-CG ($N_u = 1$)	384	416	1
ERLS-CD ($N_u = 1$)	64	128	1
ERLS-DCD ($N_u = 1$)	48	128	–
ERLS-CG ($N_u = 4$)	1392	1376	7
ERLS-CD ($N_u = 4$)	112	224	4
ERLS-DCD ($N_u = 4$)	48	224	–
ERLS-CG ($N_u = 16$)	5424	5216	31
ERLS-CD ($N_u = 16$)	304	608	16
ERLS-DCD ($N_u = 16$)	48	608	–
RLS	337	304	1
NLMS	35	35	1

TABLE XIV
COMPLEXITY OF ADAPTIVE ALGORITHMS ($N = 512$)

Algorithm	×	+	÷
ERLS-DCD ($N_u = 1$)	1536	4096	–
ERLS-DCD ($N_u = 2$)	1536	5120	–
ERLS-DCD ($N_u = 4$)	1536	7168	–
ERLS-DCD ($N_u = 8$)	1536	11264	–
RLS	264705	263680	1
NLMS	1027	1027	1

more detailed analysis of the algorithm performance will be presented in another publication. A fixed-point FPGA implementation of the exponentially weighted DCD-based RLS algorithms has also been described, which shows that the proposed algorithms are simple for finite precision implementation, require small chip resources, and exhibit numerical stability.

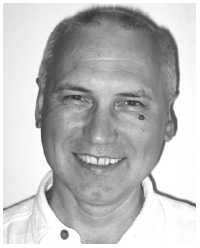
ACKNOWLEDGMENT

The authors would like to thank their colleague Dr. R. de Lamare for useful discussions of the techniques considered in this paper. The authors would also like to thank the reviewers for their comments.

REFERENCES

[1] S. Haykin, *Adaptive Filtering*, 2nd ed. Englewood Cliffs, NJ: Prentice-Hall, 1991.
[2] A. H. Sayed, *Fundamentals of Adaptive Filtering*. Hoboken, NJ: Wiley, 2003.
[3] I. D. Skidmore and I. K. Proudler, "The KaGE RLS algorithm," *IEEE Trans. Signal Process.*, vol. 51, no. 12, pp. 3094–3104, Dec. 2003.
[4] J. H. Husoy, "Adaptive filters viewed as iterative linear equation solvers," in *Lecture Notes in Computer Science: Numerical Analysis and Its Applications*. Berlin, Germany: Springer-Verlag, 2005, vol. 3401/2005, pp. 320–327.
[5] G. K. Boray and M. D. Srinath, "Conjugate gradient techniques for adaptive filtering," *IEEE Trans. Circuits Syst. I: Fundam. Theory Appl.*, vol. 39, no. 1, pp. 1–10, 1992.
[6] T. Bose and M. Q. Chen, "Conjugate gradient method in adaptive bilinear filtering," *IEEE Trans. Signal Process.*, vol. 43, no. 6, pp. 1503–1508, 1995.
[7] P. S. Chang and A. N. Willson, Jr., "Analysis of conjugate gradient algorithms for adaptive filtering," *IEEE Trans. Signal Process.*, vol. 48, no. 2, pp. 409–418, Feb. 2000.
[8] O. Diene and A. Bhaya, "Adaptive filtering algorithms designed using control Liapunov functions," *IEEE Signal Process. Lett.*, vol. 13, no. 4, pp. 224–227, 2006.
[9] T. Bose and G. F. Xu, "The Euclidean direction search algorithm in adaptive filtering," *IEICE Trans. Fundam.*, vol. E85-A, no. 3, pp. 532–539, Mar. 2002.
[10] G. F. Xu and T. Bose, "Analysis of the Euclidean direction set adaptive algorithm," in *Proc. ICASSP'98*, Seattle, WA, May 12–15, 1998, vol. 3, pp. 1689–1692.
[11] M. Q. Chen, "A direction set based algorithm for adaptive least squares problems in signal processing," in *Applied and Computational Control, Signals and Circuits: Recent Developments*. Norwell, MA: Kluwer Academic, 2001, pp. 213–236.
[12] C. E. Davila, "Line search algorithms for adaptive filtering," *Trans. Signal Process.*, vol. 41, no. 7, pp. 2490–2494, Jul. 1993.
[13] G. H. Golub and C. F. Van Loan, *Matrix Computations*, 3rd ed. Baltimore, MD: The Johns Hopkins Univ. Press, 1996.
[14] M. Al-Baali, "Descent property and global convergence of the Fletcher-Reeves method with inexact line search," *IMA J. Numer. Anal.*, vol. 5, pp. 121–124, 1985.
[15] Z. J. Shi and J. Shen, "Convergence of nonmonotone line search method," *J. Computat. Appl. Math. Elsevier*, vol. 193, pp. 397–412, 2006.
[16] S. Boyd and L. Vandenberghe, *Convex Optimization*. Cambridge, U.K.: Cambridge Univ. Press, 2006.
[17] D. K. Faddeev and B. N. Faddeev, *Numerical Methods of Linear Algebra* (in Russian). St. Petersburg, Russia: Lan, 2002.
[18] M. Fukumoto, T. Kanai, H. Kubota, and S. Tsujii, "Improvement in the stability of the BCGM-OR algorithm," *Electron. Commun. Jpn.*, vol. 83, no. 5, pt. 3, pp. 42–52, 2000.
[19] J. H. Husoy and M. S. E. Abadi, "A comparative study of some simplified RLS-type algorithms," in *Proc. 1st Int. Symp. Contr., Commun. Signal Process.*, Hammamet, Tunisia, Mar. 21–24, 2004, pp. 705–708.
[20] G. Temple, "The general theory of relaxation methods applied to linear systems," *Proc. Roy. Soc. Lond. Series A, Math. Phys. Sci.*, vol. 169, no. 939, pp. 476–500, 1939.
[21] J. H. Husoy, "RAMP: An adaptive filter with links to matching pursuits and iterative linear equation solvers," in *Proc. 2003 Int. Symp. Circuits Syst. (ISCAS'03)*, Bangkok, Thailand, May 25–28, 2003, vol. 4, pp. 381–384.
[22] Y. V. Zakharov and T. C. Tozer, "Multiplication-free iterative algorithm for LS problem," *Electron. Lett.*, vol. 40, no. 9, pp. 567–569, Apr. 2004.

[23] Y. Zakharov and F. Albu, "Coordinate descent iterations in fast affine projection algorithm," *IEEE Signal Process. Lett.*, vol. 12, no. 5, pp. 353–356, May 2005.
[24] K. Dogançay and O. Tanrikulu, "Adaptive filtering algorithms with selective partial update," *IEEE Trans. Circuits Syst. II*, vol. 48, no. 8, pp. 762–769, Aug. 2001.
[25] A. Bateman and I. Paterson-Stephens, *The DSP Handbook: Algorithms, Applications and Design Techniques*. Englewood Cliffs, NJ: Prentice-Hall, 2002.
[26] D. Boppana, K. Dhanoa, and J. Kempa, "FPGA based embedded processing architecture for the QRD-RLS algorithm," in *Proc. 12th Annu. IEEE Symp. Field-Program. Custom Computing Mach. (FCCM'04)*, Napa, CA, Apr. 20–23, 2004, pp. 330–331.
[27] J. Liu, Z. Quan, and Y. Zakharov, "Parallel FPGA implementation of DCD algorithm," in *Conf. DSP'2007*, Cardiff, U.K., Jul. 1–4, 2007, pp. 331–334.
[28] J. Liu, B. Weaver, and G. White, "FPGA implementation of the DCD algorithm," presented at the Commun. Symp., London, U.K., Sep. 2006.
[29] S. L. Gay and J. Benesty, *Acoustic Signal Processing for Telecommunication*. Norwell, MA: Kluwer Academic, 2001.



Yuriy V. Zakharov (M'01) received the M.Sc. and Ph.D. degrees in electrical engineering from the Moscow Power Engineering Institute, Moscow, Russia, in 1977 and 1983, respectively.

From 1977 to 1983, he was an Engineer with the Special Design Agency, Moscow Power Engineering Institute. From 1983 to 1999, he was the Head of Laboratory at the N. N. Andreev Acoustics Institute, Moscow. From 1994 to 1999, he was a DSP Group Leader with Nortel. Since 1999, he has been with the Communications Research Group, University of York, York, U.K., where he is currently a Reader. His interests include signal processing and communications.

George P. White received the M.Sc. degree in digital signal processing for communications from the University of Lancaster, Lancaster, U.K., in 1997 and the Ph.D. degree in optimized turbo codes for wireless channels from the University of York, York, U.K., in 2001.

From 2001 to 2007, he was with the Communications Research Group, University of York, as a Research Associate, publishing in fields such as coding and modulation, channel equalization for 3G, beamforming, high-altitude platform communications, MIMO signal processing, and modeling of amplifier nonlinearity. He is currently a Senior DSP Engineer with the Communications Division, QinetiQ, Ltd., Worcestershire, U.K.

Jie Liu received the B.S. degree in electronic science and technology from the Nanjing University, Nanjing, China, in 2004.

From 2004 to 2005, he was a Wireless Product Engineer with BenQ Co., Ltd., Suzhou, China. He is currently working toward the Ph.D. degree in FPGA design for signal processing with the Department of Electronics, University of York, York, U.K. His research interests include adaptive filtering algorithms, beamforming, and hardware design.